View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Elsevier - Publisher Connector

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 32 (2014) 928 – 934

Procedia Computer Science

The 1st International Workshop on Developing and Applying Agent Frameworks (DAAF)

An Agile Method for Multiagent Software Engineering

Jaschar Domann, Sindy Hartmann, Michael Burkhardt*, Alexander Barge, Sahin Albayrak

DAI-Labor, Technische Universität Berlin, Ernst-Reuter-Platz 7, 10587 Berlin, Germany

Abstract

The benefit of using software development methods should be a gain in regularity and clarity in software projects. Classical approaches are designed for medium to big team size with a distinctly separated and static role description combined with a top down development process. Agile methods mainly focus on small teams with low budget project size, and a fixed deadline with a bottom up development process. In this paper we present a combined approach for projects with multiple small teams growing together to a medium size project with a hard deadline and low budget – called Agile Methodology for Intelligent Agents Componentware (aMIAC). The approach will be introduced and applied within a case study.
© 2014 Published by Elsevier B.V. Open access under CC BY-NC-ND license.
Selection and Peer-review under responsibility of the Program Chairs.

Keywords: Organization, AOSE, Multi projects, Multiagent systems, agile, software engineering methodologies

1. Introduction

Managing software development is the challenge to plan activities and tasks while a software system grows in features and evolves. In the past, engineers tried to plan all activities methodically, supported by tools and frameworks. While time can be scheduled, and features have dependencies, software development is still "learning by doing". There is no difference in the challenges of handling unknown problems by developing a solution for the first-time, or by unknown expertise of the developers. Until now, it has been a Sisyphos challenge to plan the unknown.

Our approach focuses on medium sized project teams with the 'unknown' challenge in changing feature requests or unknown expertise of the developers. The whole development project is hardly time boxed with a deadline. We try to minimize the management overhead to gain efficiency in time consumption and project communication. Finally we maximize the effectiveness with a free day planning for every developer. We organize the project team similar to agent societies in the multiagent programming paradigm.

This paper is structured as follows: We give a short overview of the development models and methodologies of agent-oriented software engineering in Section 2. In Section 3 we state the gap between current software development paradigms and our problem. We present a view of managing software development and present our approach in Section 4. Finally, we explain the application of our approach by a case study in Section 5, and present the evaluation of our studies in Section 6. Additionally we wrap up with a conclusion and future work in Section 7.

* Corresponding author. Tel.: +49-(0)30-31474080 ; fax: +49-(0)30-31474003.
E-mail address: michael.burkhardt@dai-labor.de (Michael Burkhardt).

1877-0509 © 2014 Published by Elsevier B.V. Open access under CC BY-NC-ND license.
Selection and Peer-review under responsibility of the Program Chairs.
doi:10.1016/j.procs.2014.05.513

2. Standard-organisation-methods of projects and multi-projects

Nowadays the ***Waterfall model*** is considered to be one of the standard approaches for the software development process [1,2,3,4,5]. Within the Waterfall model there are distinct steps that are followed sequentially, as seen in Figure 1. The overall development process is thereby split into sub-tasks.

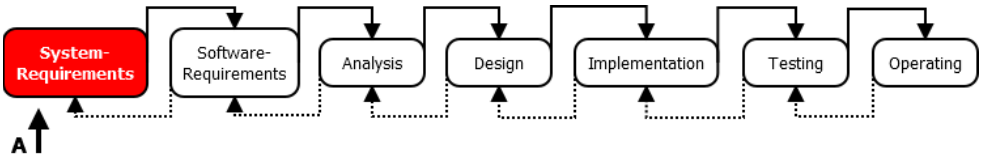

Fig. 1. Waterfall model with failure marker [1]

Each result of a sub-task has to be evaluated and approved before the developers can move on to the next task. If the developers are not able to find a solution for the current task, they have to jump back to an earlier step and re-adjust. The implementation of the overall goal therefore advances gradually with the completion of those sub-tasks.

The clear focus on the evaluation of each finished step is an advantage. A problem, on the other hand, is the missing possibility to split up the project into sub-projects and develop them simultaneously. Another major problem within this model is the inflexibility. For example, system- or software-requirements can hardly be altered during the implementation step, as seen in Figure 1. In such a case, where an adjustment is inevitable, the implementation has to be stopped and the analysis and design phase have to repeated before the implementation can be continued.

A very modern framework for software engineering is called ***Scrum*** [6,7,8]. In contrast to the Waterfall model, Scrum is based on a very incremental and iterative approach during software development. The overall project goal is split into sub-goals which are added to the product backlog. Those items within the product backlog represent the requirements that have to be implemented to build the product. The Scrum roles are based on the classic hierarchy within the company [7]. At the top of the hierarchy are the *Stakeholders*. The *Product Owner* commissions a product. He is responsible for the product and is the interface between the *Stakeholders* and the product. On the next level below the *Product Owner* is the *Scrum Master*, a kind of project manager and the connecting link between the *Product Owner* and the actual development team, which is designated as the *Scrum Team*. Depending on the size of the project to be implemented, several *Scrum Masters* and several development teams can work on one project. The actual development happens during so called sprints. Those sprints have a defined time frame in which the requirements are implemented by the developers. After a sprint is finished, the current situation is evaluated, and refinement within the product backlog is possible. The sprints are divided again into Daily Scrums, during which it is briefly discussed what was done on the previous day, where problems have occurred, and what solutions were found. In addition, a plan is made for what should be done on this day. This form of organization leads to parallelization and transparency in the work. Due to the frequent meetings a good rhythm of evaluation is possible. Changes to system or software requirements can be made without a long lead time and without a lot of overhead. Those advantages directly derive from the overall agile approach. Drawbacks are the amount of synchronization and communication needed within the hierarchy, as well as the question of how to split up certain tasks that are dependent on each other.

Within the open source programming community the organization is often loosely structured, resulting in sparse communication between the developers – the ***Xtreme programming model*** [9,10,11]. Due to the lack of organization each developer is picking his current task on his own. This may lead to the problem that more than one developer is working on the same task. This concept based on self-organization and sparse communication is commonly referred to as ”stigmergy-conception”. Every individual chose the task that seems to be the next important in the queue. By this way the group solves the overall goal without a lot of coordination.

Advantages of those methods are the clear focus on solving certain tasks and therefore implementing the overall solution without high organizational cost. However, due to the almost non-existent communication and consultation between the developers there is a high loss of economic welfare. Major problems of this organization-model are the evaluation progress and the missing synchronization between sub-teams. The latter significantly aggravates the development of sub-projects with strong dependencies. In addition, an evaluation of the results is usually only carried out by the developers themselves.

2.1. Agent-Oriented Methodologies

The *Multiagent Systems Engineering (MaSE)* is primarily a methodology for analysing, designing and developing heterogeneous multiagent systems[12,13,14]. The development process is split into two main phases. The first main phase is the analysis phase, the second is the design phase. During the analysis phase the goals are identified by user requirements and structured in a Goal Hierarchy Diagram. Afterwards Use Cases are defined and applied to create Sequence Diagrams. During the design phase, the roles are assigned to defined agent classes, the conversations between agents are constructed and the final system structure is determined.

MIAC[15,16] focuses on iterative processes that are aiming to reuse one programmed agents and multiagent systems. Therefor MIAC defines six cyclic arranged development steps. In every cycle the complete system runs through requirement management, system and user interface derivation, role modeling, implementation, integration and deployment, see figure 2. This process was extended by a store concept to enable the reuse of agents in a long running multiagent system[17].

INGENIAS is both a methodology and a set of tools for the development of Multiagent Systems (MAS)[18,19,20,21]. As a methodology it considers a MAS from five different viewpoints: organisation, agent, goals and tasks, interactions, and environment. It provides a set of concepts and relationships between the viewpoints, which enables the developer to describe the MAS. The development life-cycle process of INGENIAS initially adopts the Unified Process.

Prometheus is a methodology to develop BDI agents[22,23,24]. It includes three design phases: The *system specification phase*, the *detailed design phase* and the *architectural design phase*. The system specification phase focuses on identifying the basic functionalities of the agent system. The architectural design phase uses the functionalities of the previous phase to determine which agents the system will contain and how they will interact. The detailed design phase regards to the internals of each agent and how it will accomplish its task related to the overall system.

GAIA is a high-level, abstract agent-oriented analysis and design methodology[25,26,27,28]. The main focus of this methodology lies on the definition of roles and the implications from seeing autonomous, computational entities (the agents) as part of organizations, as in human, real-world organizations. The methodology adheres to a rigid, waterfall-like process model for the elaboration of the high-level entities and relationships. Gaia is a top-down, global view approach without noticing that the development of a MAS is still software engineering, where iterative and incremental methods are crucial success factors. But the major drawback of Gaia is its practical applicability. While Gaia produces very detailed models, they are never checked against requirements (or customers feedback) nor is the feasibility tested, for example by means of prototypes.

3. Challenges

As already mentioned, the software engineering approaches can be split into two categories, namely static and agile approaches. The agile organization models have a more iterative and incremental approach than the static ones. The final goal is also defined, but it can be adapted during development. This obviously has advantages when it comes to handling dynamically changing requirements, yet it also may cause problems concerning the evaluation and the synchronization of dependent sub-projects. Today's software has grown to be more and more complex. Those feature rich projects often times need to be addressed in a multi-project fashion to be manageable at all. A multi-project arises from a major project that can be divided into several smaller sub-projects and processed simultaneously by different companies or groups of developers. This leads to the immediate need for a way to enable and address changing and evolving requirements. The separation into sub-projects is a key success factor within distributed systems and projects, since the flawless cooperation of the sub-components is needed to achieve the desired functionality. However, this important factor is poorly addressed within existing approaches.

The Scrum and the Xtreme programming model are not ideally suited, because they do not address the complex interacting nature of multiagent systems mentioned before. The main assumption in Scrum that the complete project team is working synchronously within the same working hours does not fit. Scrum is used in small teams with one main project. Xtreme programming can handle a widely spread team, but a technical maintainer should specify single tasks. However, the agile methodologies address the possibility for changing requirements, which is an important aspect. On the other hand, the approaches based on a system development life cycle are not ideal either, since they

focus on a very sequential procedure and static project requirements engineering. But the evaluation and quality assurance is good and also an important aspect.

Therefore, we find that no approach perfectly fits and addresses the complex interacting nature of multiagent systems created by medium sized project teams or multi-project management. The closest fit is identified in the Multiagent System Engineering methodology, but it lacks the ability to easily change requirements during development. In such changes the development process would have to start from the beginning, with the Analysis and Design phase.

4. A combination of standard-organization-methods - Agile MIAC as a solution

As described above, the MIAC approach is currently the best candidate for developing multiagent systems, but it lacks agility in terms of adjusting requirements during development. We therefore propose an adaption of the MIAC approach called *Agile Methodology for Intelligent Agents Componentware*, or just aMIAC in short. In this section we describe extensions that are needed to better cope with dynamic environments.

Because a thorough goal definition and requirements management is very important for complex systems, we use an analyzing and requirement management phase of MIAC in the beginning of the project. In contrast to the original approach, this definition and specification may change during development when new insights become available. While identifying the main goals of the software project, the project team is split into developer groups to cope with the complexity of the project. Each group creates exactly one autonomous functionality – the agent – in the role modeling and implementation. Status, changes and problems are handled in a cyclic meeting with the complete project team. To decrease non-essential communication the meeting cycle should be set corresponding to the size of the project. We suggest the communication rhythm of the Scrum methodology.

We suggest not to use the strict hierarchy of the Scrum model to save organizational cost. Instead, the organization of the groups corresponds to the principle of Xtreme programming. The developer groups should be segmented into loosely coupled groups and reach a single goal. There is no need for a group leader if self-organization is working. The overall progress, however, is managed by a product manager. This role has to communicate the idea of the product to the complete project team. The following organizational steps are taken by the whole development group as shown in Figure 2.

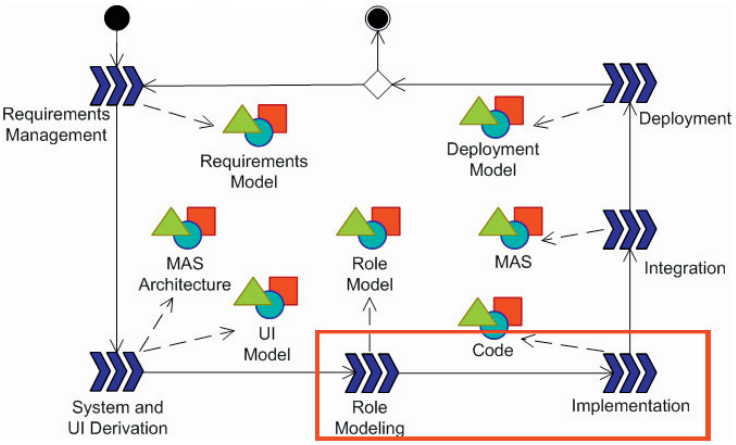

Fig. 2. MIAC methodology with the six phases [16], exhanced by parallel working developer groups

On the group level the tasks will not have to be assigned, but rather every team member can choose a task to solve the required functionality. The mechanism is similar to the Xtreme programming methodology and reduces non-essential communication in small teams. Depending on the choice of features, the developer will be presented to other groups as an expert in this topic. This way, questions can be clarified directly with the implementing person and the respective position in the interface. While role definition and implementation is done parallel in the developer

groups (red box in figure 2), the integration and deployment will be done by the complete project team. Changing requirements can make it necessary to rearrange the existing developer groups.

An advantage of the division into sub-projects is the insertion of changes in the system or software level as indicated in Figure 2 by the red box. The changes only affect a few developers, which have to go through the requirement management and system derivation phase of the MIAC approach. All other developers can continue with their work.

5. A Case Study

We apply our approach in a student course at university. The project is time boxed with a hard deadline, which is the end of the semester. All changes made must be realized and presented within this time – a span of three months. The project team – the students of the course – had various interests and are not selected by a management to realize the project. The course counts 19 students that are guided to reach the project's goal. The prototype was built in the JIAC V (Java-based Intelligent Agent Componentware V) framework[29].

The Smart Nutrition Assistant as a case study: The overall goal is to provide advisory in nutrition for a user in an ambient environment. This goal is divided into five subgoals. Every subgoal is realized by a single agent. All agents together cooperate to a multiagent system with human interaction – an assistant.

The multiagent system contains the following agents: The **Content Delivery** agent enables content providing of recipes to other agents of the system. It extends the content autonomous by retrieving recipe information from internet sources. The **Inventory Control** agent enables an inventory of all available ingredients in the household and a shopping list. A user can add purchased products with amount and expiration date. A summary of consumed products can be listed. The **Recipe Searching** agent customizes the user's search with preferences and allergies of the user. The agent filters recipes with available ingredients. If ingredients of a recipe are missing, the items can be added to the shopping list, which is managed by the *Inventory Control*. If a recipe is selected, it can be passed on to the **Cooking Assistant** agent. This agent guides the user through the preparation of meal. It shows individual cooking steps and emphasizes important instructions. After cooking the meal, ingredients were discarded from *Inventory Control*. The **Modern Interaction** component provides voice and gesture based controls, as while cooking, interacting with the tablet or keyboard is not always the best control method.

Development process description: Every agent was realized by one student group. The members of this group are selected by hand signal at the beginning of the course, in other words by the areas of interests of the students. Other proposals like expertise or profile selection were not made. This made sure that the distinct unknown assumption for the method is full-filled.

A weekly *project management meeting* contains a presentation of the current state of the agents, and informs the whole project team about status and problems. All problems can be discussed with the whole project team, so that the ideas of the other groups are included, too. This meeting was the single synchronization point for all groups and takes two hours per week. Other communication was handled via email or ticketing systems. The groups coordinated themselves if they identified an interface between two agents.

6. Evaluation

By using our method[1], the most significant time consumption was caused by the weekly project management meeting of two hours, especially considering a weekly time budget of 13 hours per developer. An average of 4 of the 12 meetings were perceived as waste of time by the developers. For most of the developers, however, the weekly project management meeting generated significant value. An average of 5 non-trivial issues were identified and solved before they became serious problems for the corresponding developer. However, with a two hours weekly meeting for a 19 person developer team, in comparison to Scrum, organizational cost was saved.

In addition to the weekly meetings for the progress presentation, there were about 20 direct communication meetings on average between developers to cope with existing issues. 100 percent of the identified problems have thereby been fixed, but on the downside, this direct communication resulted in the fact that not all of the important information

[1] All the facts used in the evaluation were obtained from a survey, which was performed with the 19 students from the case study.

was available to all developers. Due to this communication gap there were cases where confusion was created and time was lost. So, in about 2 out of 7 cases, a new problem had been created by the loss of information. In order to ensure cross-group communication without losing any information, a supporting tool was missing, but could be introduced in the future.

Through the thorough analysis and communication in the beginning of the project, most of the tasks had been identified very early within the project. There were approximately thirty mid-level features, which were divided into five groups. Due to the sub-projects the task allocation was very simple. Each developer chose three tasks from about 12 smaller tasks per week that seemed to best fit his or her abilities. This allowed the optimal use of human resources because the tasks were solved by motivated developers that are ”experts” in their field. Through this optimal use of competence, the time needed to solve a task was minimized. The only negative aspect considering the efficiency of this organization methodology is the loss of time due to meetings.

Finally this case study shows that the organization model works very well. In the end a complex functional multiagent system was created. However, it must be noted that some problems occurred by using this new model. Improvements are needed, especially in the area of communication. Also, the first evaluation of the complete project has been performed late within the overall project time-line.

7. Conclusion & Future Work

The goal of this paper was to find a suitable method for medium sized project teams that can handle the unknown challenge. We presented our approach that is adapted by multiagent system development using multi-projects where changing requirements can be implemented with reasonable effort. For this purpose different methodologies have been considered and evaluated. Since none of the existing methodologies completely suited the demand, we combined elements of those existing methodologies to form a new one. The new method was applied within a case study to show the benefits and identify weak spots.

Our research and case study have shown that there is the need for an agile software development framework for multiagent system projects. While medium sized projects are covered by classical static methodologies, agile methods focused on small team sized projects. The existing organizational models are not optimal for use within multiagent system development projects with dynamic requirements. On the one hand the static and nearly unchangeable requirements within rigid organizational models are not suitable for multiagent projects with dynamic requirements. On the other hand agile models do not comply with the structure and amount of dependencies of multiagent multi-projects.

Therefore, we propose the introduction of a new software development method for agile multiagent projects. In our model, we extended the rigid MaSE model to include an agile target and requirement definition that can develop continuously during the implementation. In addition, the communication was changed, so that the groups are briefed on the overall progress in a weekly meeting. This way every developer is informed about the progress and problems can be detected at the earliest possible stage.

However, there are also some problems that have been identified, during our case study. In some cases of a requirement change, it was not possible to inform all developers. These changes were the result of severe problems that were discovered between meetings. These problems were immediately resolved and only the relevant developers informed, since the information about it seemed irrelevant for all. The state of solved issues was not always recognized by all developers. The problems that were found during the meetings were subsequently corrected by the concerned developers. But this change of state was not always communicated back to the whole team.

To solve these problems of our organizational method, a tool is required to reduce these negative effects or even neutralize them completely. The tool is needed to add improved coordination and communication between the developers. This way all members can be informed about the implementation progress and about changes at all times, which requires a cross-project communication channel.

One possibility to gain a better overview of the problems that have occurred and the completed and outstanding tasks would be to use a bug tracker application like Redmine[2] or YouTrack[3]. With these tools, all developers could

[2] http://www.redmine.org/

[3] http://www.jetbrains.com/youtrack/

have been informed about the progress of the projects. The problem with these tools is that they do not support the loose and agile hierarchy very well, but instead depend on a strict hierarchy with managers and developers.

References

1. Royce, W.W.. Managing the development of large software systems. In: *proceedings of IEEE WESCON*; vol. 26. Los Angeles; 1970, .
2. Boehm, B.W.. A spiral model of software development and enhancement. *Computer* 1988;**21**(5):61–72.
3. McCracken, D.D., Jackson, M.A.. Life cycle concept considered harmful. *SIGSOFT Softw Eng Notes* 1982;**7**(2):29–32. URL: `http://doi.acm.org/10.1145/1005937.1005943`. doi:`10.1145/1005937.1005943`.
4. Gladden, G.R.. Stop the life-cycle, i want to get off. *SIGSOFT Softw Eng Notes* 1982;**7**(2):35–39. URL: `http://doi.acm.org/10.1145/1005937.1005945`. doi:`10.1145/1005937.1005945`.
5. Dorfman, M.. System and software requirements engineering. In: *IEEE Computer Society Press Tutorial*. IEEE Computer Society Press; 1990, p. 7–22.
6. Schwaber, K.. *Agile project management with Scrum*. O'Reilly Media, Inc.; 2004.
7. Gloger, B.. Scrum. *Produkte zuverlässig und schnell entwickeln Hanser, München* 2011;.
8. Schwaber, K.. Scrum development process. In: *Business Object Design and Implementation*. Springer; 1997, p. 117–134.
9. Beck, K., Andres, C.. *Extreme Programming Explained: Embrace Change (2Nd Edition)*. Addison-Wesley Professional; 2004. ISBN 0321278658.
10. Beck, K.. Embracing change with extreme programming. *Computer* 1999;**32**(10):70–77.
11. Kniberg, H.. Scrum and xp from the trenches. *Lulu com* 2007;.
12. DeLoach, S.A.. Multiagent Systems Engineering: A Methodology And Language for Designing Agent Systems. Tech. Rep.; DTIC Document; 1999.
13. Deloach, S.A., Wood, M.F., Sparkman, C.H.. Multiagent Systems Engineering. *International Journal of Software Engineering & Knowledge Engineering* 2001;**11**:231.
14. DeLoach, S.A.. Multiagent systems engineering of organization-based multiagent systems. In: *ACM SIGSOFT Software Engineering Notes*; vol. 30. ACM; 2005, p. 1–7.
15. Heßler, A., Hirsch, B., Küster, T.. Herding cows with JIAC v. *Annals of Mathematics and Artificial Intelligence* 2010;**59**(3-4):335–349. URL: `http://link.springer.com/article/10.1007/s10472-010-9178-x`. doi:`10.1007/s10472-010-9178-x`.
16. Heßler, A.. *MIAC: Methodology for Intelligent Agents Componentware*. Ph.D. thesis; Technical University of Berlin; 2013. URL: `http://opus4.kobv.de/opus4-tuberlin/frontdoor/index/index/docId/3610`.
17. Hessler, A., Hirsch, B., Küster, T., Albayrak, S.. AgentStore — a pragmatic approach to agent reuse. In: Dechesne, F., Hattori, H., Mors, A.t., Such, J.M., Weyns, D., Dignum, F., editors. *Advanced Agent Technology*; no. 7068 in Lecture Notes in Computer Science. Springer Berlin Heidelberg. ISBN 978-3-642-27215-8, 978-3-642-27216-5; 2012, p. 128–138. URL: `http://link.springer.com/chapter/10.1007/978-3-642-27216-5_10`.
18. Pavón, J., Gómez-Sanz, J.. Agent oriented software engineering with ingenias. In: Mařík, V., Pěchouček, M., Müller, J., editors. *Multi-Agent Systems and Applications III*; vol. 2691 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. ISBN 978-3-540-40450-7; 2003, p. 394–403. URL: `http://dx.doi.org/10.1007/3-540-45023-8_38`. doi:`10.1007/3-540-45023-8_38`.
19. Fuentes, R., Gómez-Sanz, J.J., Pavón, J.. Managing conflicts between individuals and societies in multi-agent systems. In: *Engineering Societies in the Agents World V*. Springer; 2005, p. 106–118.
20. García-Magariño, I., Gómez-Sanz, J., Pérez-Agüera, J.. A complete-computerised delphi process with a multi-agent system. In: Hindriks, K., Pokahr, A., Sardina, S., editors. *Programming Multi-Agent Systems*; vol. 5442 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. ISBN 978-3-642-03277-6; 2009, p. 120–135. URL: `http://dx.doi.org/10.1007/978-3-642-03278-3_8`.
21. Gómez-Sanz, J., Fernández-de Alba, J., Fuentes-Fernández, R.. Ambient intelligence with ingenias. In: Müller, J., Cossentino, M., editors. *Agent-Oriented Software Engineering XIII*; vol. 7852 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. ISBN 978-3-642-39865-0; 2013, p. 118–133. doi:`10.1007/978-3-642-39866-7_7`.
22. Padgham, L., Winikoff, M.. Prometheus: A pragmatic methodology for engineering intelligent agents. In: *Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies*. 2002, p. 97–108.
23. Padgham, L., Winikoff, M.. Prometheus: A practical agent-oriented methodology. *Agent-oriented methodologies* 2005;:107–135.
24. Padgham, L., Thangarajah, J., Winikoff, M.. Tool support for agent development using the prometheus methodology. In: *Quality Software, 2005.(QSIC 2005). Fifth International Conference on*. IEEE; 2005, p. 383–388.
25. Wooldridge, M., Jennings, N.R., Kinny, D.. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems* 2000;**3**(3):285–312.
26. Zambonelli, F., Jennings, N.R., Wooldridge, M.. Developing multiagent systems: The gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 2003;**12**(3):317–370.
27. Zambonelli, F., Jennings, N.R., Wooldridge, M.. Multi-agent systems as computational organizations: the gaia methodology. *Agent-oriented methodologies* 2005;**6**:136–171.
28. Cernuzzi, L., Zambonelli, F.. Dealing with adaptive multi-agent organizations in the gaia methodology. In: *Agent-Oriented Software Engineering VI*. Springer; 2006, p. 109–123.
29. Lützenberger, M., Küster, T., Konnerth, T., Thiele, A., Masuch, N., Heßler, A., et al. Engineering industrial multi-agent systems – the JIAC v approach. In: Cossentino, M., Seghrouchni, A.E.F., Winikoff, M., editors. *Proceedings of the 1st International Workshop on Engineering Multi-Agent Systems (EMAS 2013)*. 2013, p. 160–175.